SCBT-index:基于谱编码的子图索引算法

施炜杰,董一鸿,钱江波,陈华辉,辛 宇
(宁波大学信息科学与工程学院,浙江宁波 315211)

摘 要: 随着图模型规模的扩大,单机算法难以适应大规模数据集下的子图查询. 而现有的分布式算法基于无索引的简单遍历,join 过程容易出现内存溢出,而且查询图分布异常时易出现负载不均衡. 提出了一种基于谱编码的二叉索引树(SCBT-index),首先对数据图中的顶点谱编码,根据编码信息构建二叉索引树. 然后对查询图使用最小查询计划进行分解,最后 join 过程使用 3 个剪枝策略:基于拓扑结构的预剪枝、序列化 join 和基于分布式下的 join 优化. 实验结果表明,SCBT-index 在图集下的综合性能优于现有主流算法,单图下的查询时间为现有算法的 1/2 到 1/4.

关键词: 谱编码; Gini 系数; 子图查询; 子图索引
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2020)01-0110-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.01.013

SCBT-index:Subgraph Indexing Algorithm Based on Spectral Coding

SHI Wei-jie,DONG Yi-hong,QIAN Jiang-bo,CHEN Hua-hui,XIN Yu
(*Faculty of Electrical Engineering and Computer Science,Ningbo University,Ningbo,Zhejiang* 315211,*China*)

Abstract: With the expansion of graph scale,single-machine algorithms can hardly adapt to the sub-graph queries in large-scale data sets. As existing distributed algorithms are based on simple traversal without index,the join process is prone to memory overflow in the distributed algorithms and load imbalance occurs when the query graph distribution is abnormal. Therefore,a binary index tree based on spectral coding named SCBT-index is proposed. Firstly,for vertex spectrum coding in the data graph,a binary index tree is constructed according to the coding information. Then,the query graph is decomposed using the minimum query plan. Finally,three pruning strategies are used in the join process:structure matching based on topological structure,serialized join and the distributed join optimization. The experimental results show the comprehensive performance of SCBT-index under the graph set is better than that of the popular algorithms. In addition,the query time under the single graph is 1/2 to 1/4 of that of the existing algorithms.

Key words: spectral coding;Gini;subgraph query;subgraph index

1 引言

现实社会中,图被广泛应用于表示实体与实体之间的关系. 子图查询作为最基本的图操作应用于各个行业,如社交网络、生物分析、人才推荐、金融反欺诈等领域[1]. 子图查询是给定一个查询图,找出图集或大图中所有与查询图同构的子图,是一个 NP-hard 问题[2]. 现有主流算法归纳为两种模式:一种是无过滤验证模式(NFV,No-Filter Verify)[3],以匹配为主的策略,将查询图分解成各个子结构,通过集合运行以及 join 得到答案集. 如 GraphQL[4] 算法以半径 r 为搜索空间,索引数据集中的所有顶点,获取每个节点的所有匹配结果;sPath[5] 在 GraphQL 的基础上增加了邻域签名,匹配查询图中的路径与数据图中路径. disHHK[6] 算法是首个分布式查询算法,包括调度和匹配两部分:调度为匹配后的 join 操作提供策略,匹配则根据 DFS 原则展开. STwig[7] 算法将查询图分解的方法进行并行化处理,但在 join 过程中容易出现大量冗余结果以及负载不均衡等问题. 另一种是过滤加验证模式(FTV,Filter-Then-Verify)[3],通过对数据图模式分析以及用户行为习惯的发现构建过滤模型,需经历索引构建、查询、验证三个阶段. Lei Zou 等人提出的 gCode[8] 算法将图的拓扑结构作为编码构建索引,通过编码的包含关系完成查询中的匹配. CT-index[9] 通过对图中路径、树和简单环的枚举,使用 fingerprint 技术将各个特征映射至不同的桶构建索引. Philip S Yu 团队提出的 gIndex[10] 首次将模式挖掘

收稿日期:2018-11-06;修回日期:2019-07-29;责任编辑:李勇锋
基金项目:国家自然科学基金(No.61572266,No.61602133);浙江省自然科学基金(No.LY20F020009,No.LZ20F020001)

应用于索引构建,采用增量式判别函数对特征进行动态选择构建索引树. 作为 gIndex 算法的改进, FG-index[11]算法对特征进行二次筛选,同时保存冗余和非冗余特征以构建倒排索引. Lindex[12]算法依据特征之间的包含关系构建索引,采用图点阵作为索引结构,优化索引的构建时间和查询速度.

NFV 模式通过顶点匹配的形式进行查询,泛化能力强,能在图集和单图完成任务,但该模式索引过滤性能差,匹配中存在大量的冗余结果,计算时耗大,分布式环境下易出现大量的 join 操作以及负载不均衡等问题. 而 FTV 模式以包含关系进行过滤,主要适用于图集上的查询任务,过滤性能好,查询快捷,但该模式在构建前期都会涉及到复杂的模式发现问题,构建时耗大,重构代价大,泛化性能差,只适用于特定领域下的查询任务. 针对这些缺陷,本文提出一种分布式下基于谱编码的二叉索引树的子图索引算法(Spectral Coding Binary Tree index,SCBT-index),主要贡献如下:(1)提出基于谱编码的子图查询索引算法 SCBT-index,将数据图顶点进行谱编码,使用 Gini 系数构建二叉索引树;(2)提出最小查询算法减少单个查询图的查询次数,通过基于拓扑结构的 join、序列化 join 等优化策略降低算法的整体计算时间;(3)实验表明 SCBT-index 对结构相似的节点有着极好的聚类效果,能完成图集和单图的查询任务,泛化性能强,在查询准确性不变的前提下,图集中的综合表现优于已有算法,单图中的查询时间为现有算法的 1/4 到 1/2.

2　基本概念

标签图 $G(V,E,L)$,其中 $V=\{v_i\}$,$i=1,2,3,\cdots,n$ 为顶点集,$E\subseteq V\times V$ 为边集,L 为映射函数,$L(v_j,v_k)=e,e\in E$.

定义 1　图同构[13]:图 $G=(V,E,L)$ 和图 $G'=(V',E',L')$,当且仅当图 G 和 G' 间存在一一映射 I,使得 $V\rightarrow V'$,同时存在边 $L(v,u)\in E$,有 $I(v)=v'$, $I(u)=u'$, $L'(v',u')\in E'$ 时,称图 G 和 G' 同构,记为 $G\cong G'$.

定义 2　子图同构[13]:图 $G=(V,E,L)$ 和图 $G'=(V',E',L')$,当且仅当图 G 和 G' 间存在单项映射 I,使得 $V\rightarrow V'$,同时存在边 $L(v,u)\in E$,使得 $I(v)=v'$, $I(u)=u'$, $L'(v',u')\in E'$ 时,称图 G 子图同构于 G',记为 $G\cong G'$.

定义 3　子图查询[2]:给定一个图集 $D=\{g_i\}$,$i=1,2,3,\cdots,n$ 或一个单图 G 和一个查询图 G_q,返回图集 G_A,使得 $G_A=\{g\mid M(G_q,g),g\in D\}$ 或 $G_A=\{g\mid M(G_q,g),g\in G\}$ 函数 M 为匹配函数,返回布尔值{0,1},其中当 $G_q\cong g$ 时返回 1.

定义 4　顶点生成树图 $G(V,E)$ 中,以任意顶点 v 为根节点进行 BFS(宽度优先遍历),生成以 v 为根节点的 n 层生成树 $T_n(G,v)$,称为顶点 v 的 n 阶生成树.

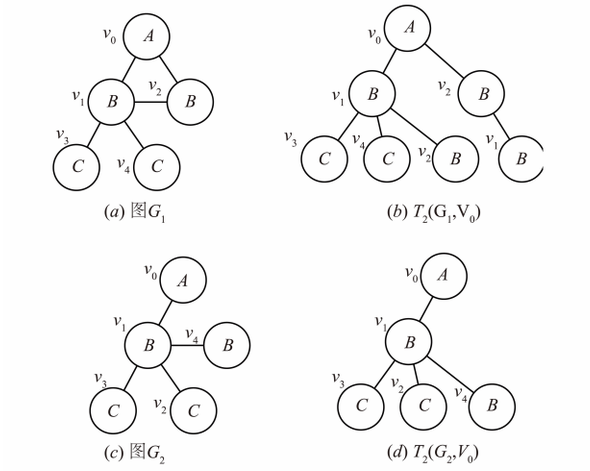


图1　顶点n阶树的生成示意图

图 1 为顶点的 2 阶生成树的示意图,其中图 1(*b*)是图 G_1 中以顶点 v_0 为根节点进行 BFS 遍历,当深度为 2 时,可得图 1(*b*)的生成树 $T_2(G_1,v_0)$;图 1(*d*)为图 G_2 中以顶点 v_0 为根节点进行 BFS 遍历,当深度为 2 时,图 1(*c*)的生成树 $T_2(G_2,v_0)$.

3　SCBT-index

对于图集数据集,SCBT-index 算法将每个数据图视为一个实体构建索引. 因此在分布式计算中,各个节点上的计算结果不会相互影响,只需要在数据划分时考虑到各个分区的负载问题即可. 对于单图数据集,算法以密度相近的原则进行数据划分,并针对每个 RDD 构建索引,考虑负载均衡和跨节点子图等情况.

SCBT-index 算法首先对图中的顶点进行谱编码,然后使用 Gini 系数构建二叉索引树,索引树将结构相同的子图片段存储在同一个叶子节点下. 查询过程中首先将查询图分解成较小的查询子单元,并根据各个子单元在索引树中的匹配结果返回答案集,在 join 过程中通过序列化 join 和在分布式下的 join 减少时空开销. 为防止边遗失的情况,算法在查询结束后对各个节点下的候选顶点集按照顶点的 join 序列进行重分区.

算法包括 offline 和 online 两部分,如图 2 所示. offline 主要包括顶点的谱编码和索引构建两个阶段. 在数据预处理阶段读取图数据,获取每个顶点的谱编码;索引构建阶段根据数据集中每个顶点编码构建二叉索引树,保证系统下索引树的平均深度最小. online 部分为查询操作,根据索引对查询图的顶点编码获取数据图中的匹配顶点,最后得到答案集.

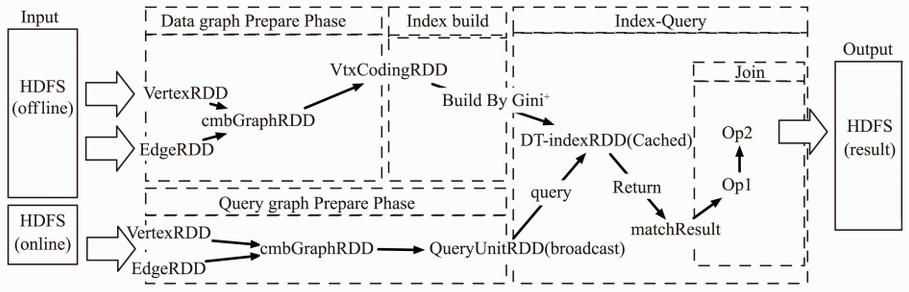


图2 SCBT-index 算法流程框架图

3.1 顶点的谱编码

谱编码是 SCBT-index 算法的第一步,通过对顶点结构的编码可以有效提高子图查询的匹配速度. 首先获取顶点标签在全局下的 onehot 编码. 假设图 G 的顶点有 p 个标签,顶点 v 的谱编码由 $<L,S(n),Eig(m)>$ 三元组构成. $L=(l_1,l_2,\cdots,l_p)$ 为顶点 v 的标签在全局范围内的 onehot 编码,记为 $v.L$.

定义 5 同胚节点 给定两个图 G 和 G',顶点 $v\in G,v'\in G'$. 顶点谱编码为 $<L,S(n),Eig(m)>$ 和 $<L',S'(n),Eig'(m)>$,如果顶点编码满足 $\forall i,l_i=l_i',s_i\leqslant s_i'$, $i\in[1,p]$, $\forall j,\lambda_j\leqslant\lambda_j',j\in[1,k]$ 则称 v 为 v' 的同胚节点. 如图 1,图 G_2 的顶点 v_0 是图 G_1 的顶点 v_0 的同胚节点.

引理 1[8] 图 G 和 G',顶点 $v\in G,v'\in G'$,记 v 为中心生成的邻接矩阵的特征值为 $(\lambda_1,\lambda_2,\cdots,\lambda_k)$,以 v' 为中心生成的邻接矩阵的特征值为 $(\beta_1,\beta_2,\cdots,\beta_t)$,当 $G\backsimeq G'$ 时,则 $\forall i,\lambda_i\leqslant\beta_i,i\in[1,k],k\leqslant t$.

定理 1 现有图 G 和 G',顶点 $v\in G$,如果 G' 中不存在顶点 v 的同胚节点,则图 $G\backsimeq G'$.

证明 反证法,证明略.

3.2 索引构建

SCBT-index 使用基尼系数选择特征构建二叉索引树,通过基尼系数计算第 i 类样本在总样本数据中“纯度”以此来选择二叉索引树的分割点. 根据顶点编码和分类要求如下定义 $gini(x)$:

$$gini(x)=1-\left(\frac{|C_i|}{|A|}\right)^2 \tag{1}$$

$|C_x|$ 为在分割点 x 下的样本数量. 最后特征 x 在集合 A 下的 $Gini$ 系数计算公式:

$$Gini(A,x)=\frac{|C_{x1}|}{|A|}gini(x_1)+\frac{|C_{x2}|}{|A|}gini(x_2) \tag{2}$$

其中 $|C_{x1}|,|C_{x2}|$ 分别为将 x 作为分割点对 A 分割后两个样本集,即左右子树的顶点集,$|C_{x1}|+|C_{x2}|=|A|$.

定理 2 对 n 个顶点的谱编码集 $A=\{<L,S(n),Eig(m)>\}$ 构建二叉索引树,其中 $Gini(A,x)$ 为特征 x 在取不同值下的基尼值,那么,将 $\max|Gini(A,x)|$ 作为节点的分裂条件,得到的索引树查询复杂度为 $O(\log_2 n)$.

证明 $\because |C_{x1}|+|C_{x2}|=|A|$,

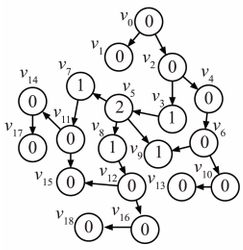
$$\begin{aligned}Gini(A,x)&=\frac{|C_{x1}|}{|A|}gini(x_1)+\frac{|C_{x2}|}{|A|}gini(x_2)\\&=\frac{|C_{x1}|}{|A|}\left(1-\left(\frac{|C_{x1}|}{|A|}\right)^2\right)\\&\quad+\frac{|C_{x2}|}{|A|}\left(1-\left(\frac{|C_{x2}|}{|A|}\right)^2\right)\\&=1-\frac{|C_{x1}|^3+|C_{x2}|^3}{|A|^3}=\frac{3\cdot|C_{x1}|\cdot|C_{x2}|}{|A|^2}\\&=\frac{3}{4}-3\left(\frac{|C_{x1}|}{|A|}-\frac{1}{2}\right)^2\end{aligned}$$

当 $C_{x1}=\lfloor\frac{A}{2}\rfloor$ 时 $Gini(A,x)$ 取最大值,进行分裂时左右子树的节点数量相等或差为 1,以此类推得到的二叉索引树是平衡树. 因此 n 个顶点构成的二叉索引树的深度 h 满足 $2^{h-1}-1<n\leqslant 2^h-1$,取对数得到 $h-1<\log_2 n+1\leqslant h$,$\therefore h=\lfloor\log_2 n\rfloor+1$.

查询时,假设有查询概率均为 $P_i=\frac{1}{n}$ 的 n 个叶子节点,索引树的平均查询长度 $ASL=\sum_{i=1}^{n}P_i\cdot C_i$,$C_i$ 为各个叶子结点到根节点的路径长度,可得 $ASL=\sum_{i=1}^{n}P_i\cdot C_i\leqslant\frac{1}{n}\sum_{i=1}^{n}(\log_2 n+1)\leqslant\log_2 n+1$. 综上查询时间复杂度为 $O(\log_2 n)$. 证毕.

如图 3(a)中的图 G,可得到图 3(b)的顶点编码,各个顶点的 $Gini$ 系数如表 1. 根据定理 2,表 1 中当特征 λ_1 为 1 时,$Gini(A,\lambda_1=1)=0.7456$ 最大,因此构建 G 的索引时首先选择 $\lambda_1=1$ 作为根节点的分裂特征值. 以此递归计算,直到节点所包含的顶点特征值都相等,最终得到图 4 的索引树.

索引构建以深度优先的原则递归选择分裂特征构建二叉索引树,伪代码如算法 1.



(*a*) 图*G*

ID	l_1	l_2	l_3	S_1	S_2	S_3	λ_1	λ_2
0	1	0	0	2	0	0	1.41	0
2	1	0	0	1	1	0	1.41	0
3	0	1	0	0	0	1	1	0
4	1	0	0	1	0	0	1	0
5	0	0	1	0	3	0	1.73	0
6	1	0	0	1	1	0	1.41	0
7	0	1	0	1	0	0	1	0
8	0	1	0	1	0	0	1	0
10	0	1	0	1	0	0	1	0
11	1	0	0	2	0	0	1.41	0
12	1	0	0	2	0	0	1.41	0
14	1	0	0	1	0	0	1	0
16	1	0	0	1	0	0	1	0

(*b*) *G*的顶点编码

图3　图*G*的顶点编码

表 1　各个特征下在图 *G* 下的 Gini 系数

分裂特征	分裂值	Gini 系数
l_1	0	0.7101
l_2	0	0.6391
l_3	0	0.2130
s_1	1	0.5325
s_1	0	0.3905
s_2	1	0.5325
s_3	1	0.2130
λ_1	1	0.7456
λ_1	1.41	0.2130

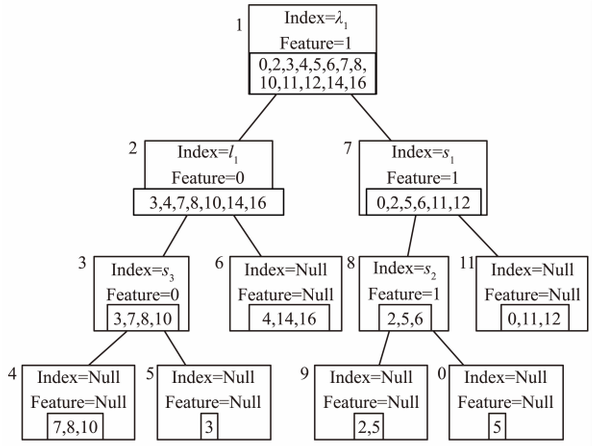


图4　*G*的索引树

算法 1　图索引的构建 Build_index(vtxCode)

输入:vtxCode 为顶点编码;
输出:SCBT-index.

```
1.  def split( vtxCode,parent):  //节点分裂函数
2.      if stop(vtxCode) return leaf//停止分裂判定
3.      val right_code,left_code,split_value = get_best_split(vtxCode)
//根据 Gini 系数选择分裂点
4.      val root = new parent(parent)
5.      root. split_value = split_value
6.      root. left = split(left_code,root)
7.      root. right = split(right_code,root)
8.  return index
```

3.3　查询

通过生成最小查询计划减少查询次数降低查询时间,然后 join 得到的顶点候选集返回最后的答案集.

给定一个查询图 Q,依次要对图中顶点 $v \in Q$ 做匹配计算,对于顶点数为 n 的查询图 Q 生成的顶点候选集大小为 $\sum_{i=1}^{n} |v_i.match|$,候选集大小决定 join 的时空开销.

图 5 展示了查询图 Q_1 的多种分解策略,不同的分解产生的子结构数量也不同. 匹配过程中,从结构的复杂度和数量关系出发,分解图 q_1 和 q_2. 匹配时 $q_1(1)$ 顶点候选集明显多于 $q_2(1)$. 从查询整体匹配结果出发,3 个子图单元的分解图 q_2 整体的顶点候选集小于分解图 q_1.

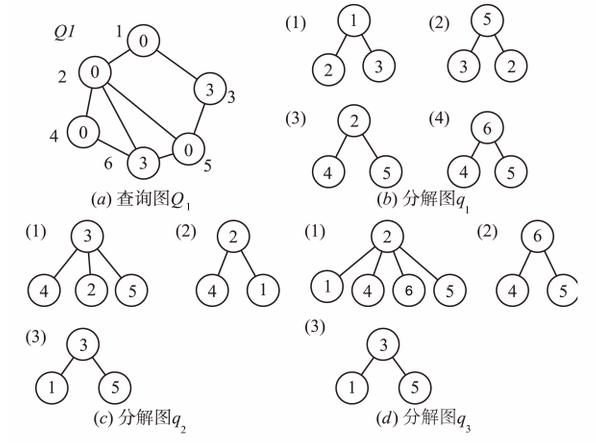


图5　查询图分解示意图

定义 6　图的最小查询　给定一个图 $G=(V,E)$,给定集合 $L=\{Sub_1,Sub_2,\cdots,Sub_i,\cdots,Sub_n\}$,其中 $Sub_i=(V_i,E_i)$ 为 G 的分解子单元,满足 $\cup_{i=1}^{n} E_i = E$, $\forall S_i,S_j \in L, i \neq j, E_i \cap E_j = \varnothing$. $\min|L|$称为图 G 的最小查询.

图的最小查询采用近似算法:首先获取图中度最大的顶点 v 和 v 的所有邻接顶点集 U,并删除图中边 (v,U),重复该过程直到所有边被移除. 算法伪代码如算法 2.

算法 2　图的最小查询生成算法 MinQuery(vtxRdd,edgeRDD)

输入:vtxRdd 为顶点的分布式下存储单位,edgeRDD 为边的分布式下存储单位;
输出:最小查询单元 L.

```
1. var queryGraph = graph( vtxRdd,edgeRDD )
2. val queryUnit = ArrayBuffer( )
3. while queryGraph. edge is not empty:
4.      val vtx = queryGraph . get_max_degree( )
5.      val vtx_child = queryGraph. get_ child(vtx)
```

```
6.    for (child < -vtx_child) queryGraph . dropedge(vtx,child)
7.    queryUnit + = (vtx - > vtx_child)
8. return queryUnit
```

查询图分解后,算法通过索引获取各个子结构在数据图下的候选顶点集. 查询过程遵从定理1,过滤不满足同胚节点条件的顶点. 如将图3中的顶点v_1作为查询顶点,顶点编码<(1,0,0),(1,0,0),(1.0,0.0)>,根据图4中G的索引进行查询. 查询步骤如下:1. 查询从根节点开始,判断节点的分裂特征为λ_1,大小为1,对应v_1. $\lambda_1 \leqslant 1$,跳转至索引节点;2. 根据节点的分裂特征信息跳转至节点6,为叶节点,停止查询,最后返回匹配结果. 因此,查询节点v_1在G中的候选匹配顶点集为$\{v_4, v_{14}, v_{16}\}$.

3.4 join 操作

使用基于图的拓扑结构的join提出三个剪枝策略对中间结果进行预剪枝.

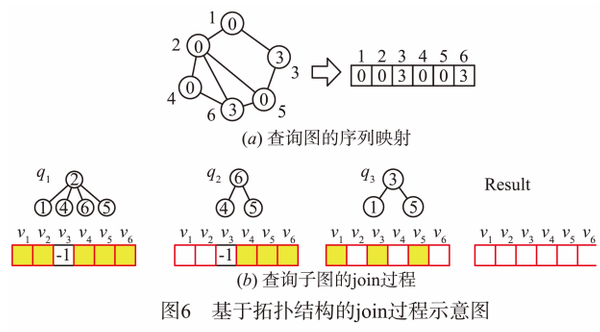
(1)基于拓扑结构的预剪枝. 首先确定join的起始节点,通过根节点的拓扑扩展确定后续匹配节点,直到得到与查询图结构相同的匹配图. 以图6为例,对于顶点u_1, u_2, u_3,候选顶点集分别为$\{v_1, v_2, v_3, v_4, v_5, v_6\}$,$\{v_1, v_2, v_3, v_4, v_5, v_6\}$,$\{v_1, v_2, v_3, v_4, v_5, v_6\}$. 以$< u_1, u_2, u_3 >$为join序列,假设以$u_1$为根节点,确定$\{v_1, v_2, v_3, v_4, v_5, v_6\}$为匹配根节点,然后对于匹配根节点$v_1$,通过数据图的拓扑结构确定$v_6, v_3$为扩张节点,因此对于以$v_1$为根节点的扩张后匹配结果为$\{v_1, v_3\}$,$\{v_1, v_6\}$,然后再进行扩张,得到最后答案为$\{v_1, v_6, v_4\}$,$\{v_1, v_3, v_4\}$,$\{v_1, v_3, v_2\}$,$\{v_1, v_3, v_5\}$. 在join过程中避免了产生类似于$\{v_1, v_2\}$,$\{v_1, v_4\}$,$\{v_1, v_5\}$等中间结果,减少了计算的资源消耗. 因此针对join过程有以下策略:①序列化映射查询图;②将匹配的各个子查询图映射至查询图序列中;③重复2过程,直接删除中间不匹配的结果. 如图6(a)中首先将查询图Q进行序列映射生成右边的查询序列,图6(b)为查询子图的join过程,针对查询图Q分解后的q_1, q_2, q_3,3个子图依次将匹配结果映射至查询序列中,如果节点标签不匹配,直接剪枝掉这个中间结果,不再进行join操作.

(2)join序列的生成. 确定拓扑扩张的根节点,不同的根节点join的计算代价不同. 使用以下两个规则对各个匹配节点进行排序:

(Ⅰ)优先选择结构复杂的节点;

(Ⅱ)优先选择频繁度低的节点;

规则(Ⅰ)保证一次join固定尽可能多的节点,规则(Ⅱ)保证一次join产生尽可能少的中间结果. 根据函数$value(x) = \dfrac{\deg(v)}{|v.match|}$计算顶点$value$,对顶点降序排序生成join序列.



(a) 查询图的序列映射

(b) 查询子图的join过程

图6 基于拓扑结构的join过程示意图

(3)分布式下的join优化

对各个分解子图的匹配结果进行聚合分类生成新的RDD,再根据$value(x)$得到join序列. 过程中,只本地化当前序列下的匹配RDD降低内存的使用开销.

3.5 负载均衡

分布式下的算法运行呈现木桶效应,需要对各个worker下的数据量分布进行优化,以实现负载均衡,达到算法整体的最优计算时间. 主要采用sorted-greedy[14]的方法来对数据进行重新划分,将待join的顶点候选集按照大小进行降序排序,逐个将各个分区的匹配结果以最小单位从数据域大的分区放置数据域小的分区.

4 实验结果与分析

实验使用scala2.10.4开发,集群环境为Spark2.1.0,集群由一台master和13台Slave组成,所有节点都是PC机,运行时使用的通用参数为executor-memory为4G. executors和partition数量在实验对比中根据对比条件和实验效果动态调整. 实验使用表2中6个真实的数据集进行实验对比. 查询图Q根据数据集随机生成,实验保证两个规则下的查询图数量相等.

表2 实验数据集描述

Data Type	Dataset	Vertex(ave)	Edge(ave)	Vertex label	Edge label
Graph Set	AIDS	24.80	26.80	62	3
	YouTube	16264.27	42720.61	47	1
Big graph	WordNet	82670	133445	5	1
	Amazon	262111	1234877	100	1
	Hyves	1402611	2777419	70	1
	USPatents	3774768	16522438	418	1

对比实验主要分为图集下的查询速度对比和单图下的查询速度对比,不同数据类型下的对比算法不同,实验对比算法为图集下的Lindex[12]和CT-index[9]算法,单图下的Stwig[7]和TurboISO[15]算法.

4.1 算法参数分析

参数n和m分别是SCBT-index在编码时$S(n)$和$Eig(m)$的生成树深度,$S(n)$主要保留了邻居的标签信

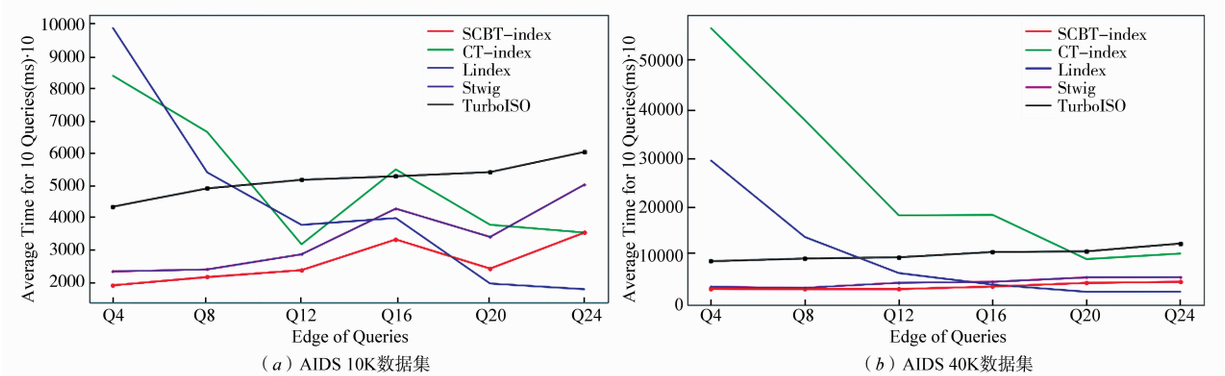
息,$Eig(m)$保留了顶点的拓扑结构信息,不同大小的生成树影响编码剪枝力. 表 3 是 Amazon 数据集上进行的剪枝力对比实验,使用 Q4 作为查询集. 剪枝力的计算公式如下:$pruningRate = 1 - |v.match|/|v.label|$,其中$|v.match|$为顶点的在参数下的匹配数量,$|v.label|$为在数据集下与该顶点具有相同标签的顶点数量. 可见随着 n 或者 m 的增大编码剪枝力也逐渐上升.

表 3　参数 n 和 m 的剪枝力对比实验

	$n=1$	$n=2$	$n=3$
$m=1$	0.6615	0.6688	0.6765
$m=2$	0.6692	0.6727	0.6750
$m=3$	0.6769	0.6808	0.6843

4.2　算法运行时间

图 7 是在 AIDS10K 和 AIDS40K 两个数据集上通过 Q4,Q8,Q12,Q16,Q20,Q24 六个查询集进行算法性能对比. 由于 Lindex,CT-index,TurboISO 算法是单机算法,因此为了保证实验的公平性,实验在平台内存的一致性前提下,对 SCBT-index,Stwig 的运行采用 Local 模式[2],partition 为 2 的运行方案. 随着查询集的增加,特征驱动的 Lindex 和 CT-index 算法查询时间逐渐减少,而 SCBT-index,Stwig,TurboISO 的查询时间逐渐增加,并伴有小幅波动. 在 Q4 到 Q16 之间,SCBT-index 的查询时间均要小于其他算法,在 Q20 之后 SCBT-index 的查询时间要次于 Lindex,主要因为 Lindex 是基于频繁子图构建的索引,较大的查询图意味更多的剪枝特征,增强了算法的剪枝能力. 综上, CT-index 的平均查询时间最大,Lindex 次之,SCBT-index 最小. 图 8 为 YouTube 数据集下的实验结果,其数据规模远大于 AIDS 数据集,Lindex 和 CT-index 未在 12 小时内完成查询任务,从综合性能来看,SCBT-index 的性能要优于其他算法.



(a) AIDS 10K数据集　　(b) AIDS 40K数据集

图7　AIDS 数据集的性能对比

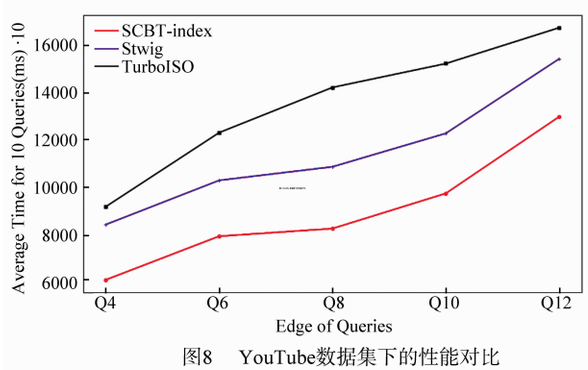


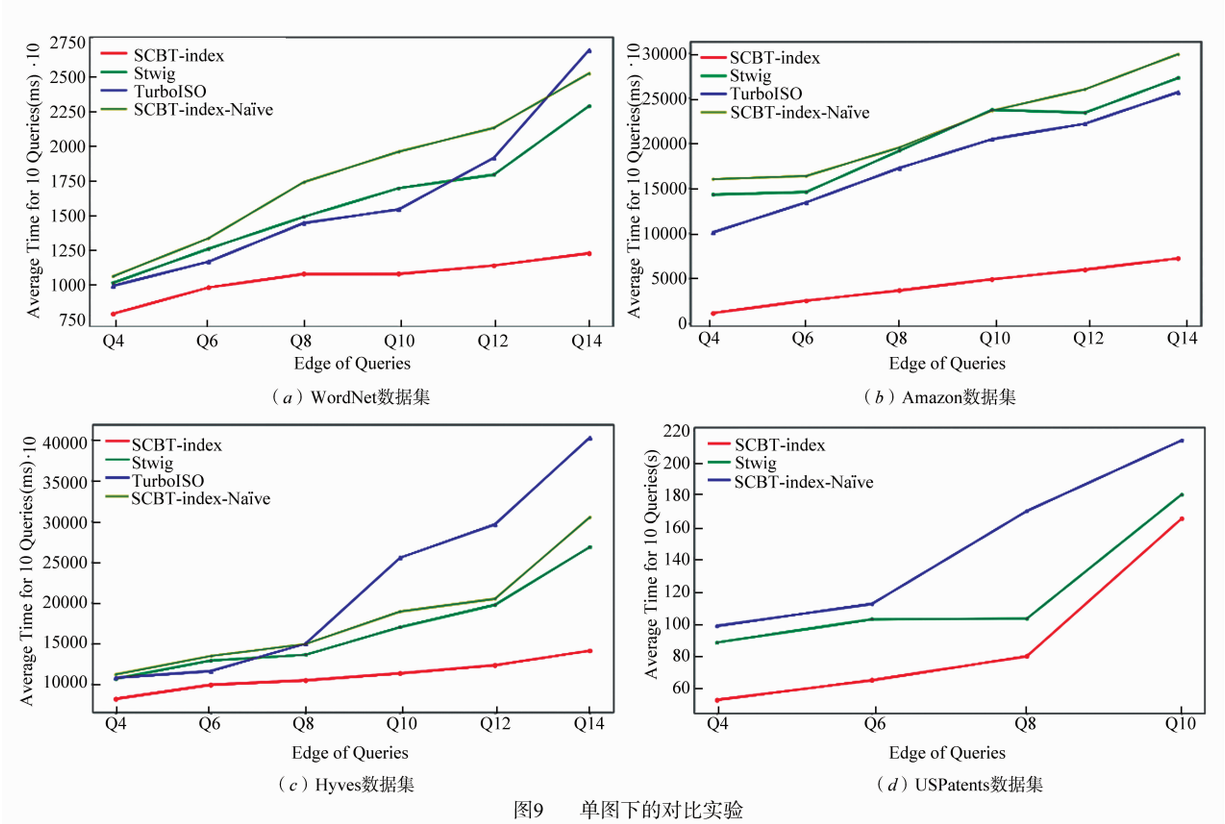
图8　YouTube数据集下的性能对比

图 9 是单图下的查询时间对比. 将 TurboISO 移植到分布式平台下,SCBT-index-naïve 为只 join 过程保留基于拓扑结构. 实验参数 executor-memory = 5GB,executor-num = 13. 图 9 显示 SCBT-index 的查询时间优于其他算法,在图 9(a)中当数据集较小时算法性能差距较小, SCBT-index 的查询速度大约是 Stwig 算法 1.5 ~ 2 倍之间. 随着数据集密度的增大,单点结构较为复杂时,如图 9(b), SCBT-index 查询时间最短,因为采用了拓扑扩展的剪枝策略,以及 join 序列和分布式下的 join 优化. 在图 9(c)中,预处理阶段已去除度大于 500 的顶点, TurboISO 算法的平均查询时间最长,SCBT-index-naïve 次之,SCBT-index 时间最优,这是因为 TurboISO 算法根据邻居特征进行剪枝的策略失去了优势,未能过滤掉大量的无用匹配结果. 图 9(d)是千万级稀疏图下的性能效果,TurboISO 算法未能在 12 小时内返回结果,SCBT-index-naïve 由于未采用序列化 join 其查询时间最长,SCBT-index 和 Stwig 算法时耗趋势相近.

4.3　并行度实验

表 4 和表 5 是不同数据集下 SCBT-index 算法并行度对比实验,其中表 4 是 salve = 13 时不同分区数下对比实验,表 5 是在不同并行度下的实验. 表 4 显示 WordNet 和 Amazon 数据集下,分区数量在 75 到 125 时 SCBT-index 的查询时间最佳,此时数据集的分片刚好

满足平台中 core 的 1 到 1.5 倍左右的数量,能保证平台中 CPU 的利用率达到最好,但 Hyves 数据集下最佳并行度在 175 左右,因为数据量过大时每个分区计算量也随之增大,导致部分分区的计算时间较长,造成队列整体等待时间过长;表 5 中 wordNet 和 Hyves 数据集下数据较稀疏,查询图在数据图中分布较均匀,随着 Parallel 数量的增加查询时间逐渐减少,同时减少趋势变缓;Amazon 数据集下则是先减小后增加的趋势. 这主要与查询图在数据图中分布有关,当数据集中顶点的度分布不均匀时,会导致部分查询结果大量分布同一个节点,增加了节点间的通讯成本. 因此,SCBT-index 算法对分区数和并行度的选择需要根据具体的数据集,从数据集密度、大小以及查询图的分布等情况综合考虑来选择最优的区间.



(a) WordNet数据集 (b) Amazon数据集

(c) Hyves数据集 (d) USPatents数据集

图9 单图下的对比实验

表 4 不同分区数下运行时间(单位:ms)

分区数	25	50	75	100	125	150	175	200
WordNet	2536	2264	2080	2141	2234	2193	2286	2497
Amazon	3770	3465	2989	3094	3000	3297	3403	3322
Hyves	9011	8776	8479	8653	8362	8320	8184	8322

表 5 不同集群数量下运行时间(单位:ms)

并行度	1	3	5	7	9	11	13
WordNet	3260	3233	3057	3057	3028	3049	3090
Amazon	2928	2682	2745	2884	2965	3079	3329
Hyves	10485	9399	9065	9089	8986	8944	8932

5 总结

提出了一种基于 Spark 的图索引查询算法 SCBT-index,使用 Gini 系数对图中的顶点谱编码进行分类、构建索引. 然后提出基于拓扑结构的预剪枝、序列化 join 和基于分布式下的 join 优化三个优化策略,并根据分布式下的计算环境优化了 join 过程,降低了 join 的时空开销. 最后实验表明 SCBT-index 在图集中的综合表现均好于 Lindex 和 CT-index,面对数据的横向和纵向扩张时都具有良好的适应能力;在单图中查询速度是 Stwig 和 TurboISO 算法的 1/2 到 1/4,降低了大图下子图查询的时间复杂度.

参考文献

[1] Lappas T, Liu K, Terzi E. Finding a team of experts in social networks[A]. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining[C]. Paris: ACM, 2009. 467 – 476.

[2] Katsarou F, Ntarmos N, Triantafillou P. Performance and scalability of indexed subgraph query processing methods

[A]. Proceedings of the VIDB Endowment[C]. Hawaii, 2015,8(12):1566 – 1577.
[3] Liang R, Hai Z, Jiang X, et al. Scaling hop-based reachability indexing for fast graph pattern query processing[J]. IEEE Transactions on Knowledge & Data Engineering, 2014,26(11):2803 – 2817.
[4] H He, A K Singh. Graphs-at-a-time:query language and access methods for graph databases[A]. Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data[C]. New York:ACM,2008. 405 – 418.
[5] S Zhang, S Li, J Yang. GADDI:Distance index based subgraph matching in biological networks[A]. Proceedings of the 12th International Conference on Extending Database Technology:Advances in Database Technology[C]. New York:ACM,2009. 192 – 203.
[6] Ma S, Cao Y, Huai J, et al. Distributed graph pattern matching[A]. Proceedings of the 21st International Conference on World Wide Web[C]. New York: ACM, 2012. 949 – 958.
[7] Zhao Sun, Hongzhi Wang, Haixun Wang, et al. Efficient subgraph matching on billion node graphs[A]. Proceedings of the VLDB Endowment[C]. Istanbul,2012,5(9):788 – 799.
[8] Zou L, Chen L, Yu J X, et al. A novel spectral coding in a large graph database[A]. Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology[C]. New York: ACM, 2008. 181 – 192.
[9] Y Xie, P Yu. CP-Index:on the efficient indexing of large graphs[A]. Proceedings of the 20th ACM International Conference on Information and Knowledge Management[C]. New York:ACM,2011. 1795 – 1804.
[10] Yan X, Yu P S, Han J. Graph indexing:a frequent structure-based approach[A]. ACM SIGMOD International Conference on Management of Data[C]. New York: ACM,2004. 335 – 346.
[11] Cheng J, Ke Y, Ng W, et al. Fg-index:towards verification-free query processing on graph databases[A]. Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data[C]. New York: ACM, 2007. 857 – 872.
[12] D Yuan, P Mitra. Lindex:a lattice-based index for graph databases[J]. VLDB Journal,2013,22(2):229 – 252.
[13] H Shang, Y Zhang, X Lin, J X Yu. Taming verification hardness:an efficient algorithm for testing subgraph isomorphism[J]. PVLDB,2008,1(1):364 – 375.
[14] 严玉良,董一鸿,何贤芒,等. FSMBUS:一种基于 Spark 的大规模频繁子图挖掘算法[J]. 计算机研究与发展, 2015,52(8):1768 – 1783.
[15] WS Han, J Lee, JH Lee. Turboiso:towards ultrafast and robust subgraph isomorphism search in large graph databases[A]. Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data[C]. New York:ACM,2013. 337 – 348.

作者简介

施炜杰　男,1993 年生. CCF 学生会员,2019 年获宁波大学计算机技术专业硕士学位,主要研究方向为大数据、数据挖掘.

董一鸿(通讯作者)　男,1969 年生于浙江宁波. 博士,CCF 会员,宁波大学教授,主要研究方向为大数据、数据挖掘和人工智能等.
E-mail:dongyihong@ nbu. edu. cn

陈华辉　男,1964 年生于浙江宁波. 博士,CCF 会员,宁波大学教授,主要研究方向为数据库技术、流数据处理.
E-mail:chenhuahui@ nbu. edu. cn

钱江波　男,1974 年生于浙江宁波. 博士,CCF 会员,宁波大学教授,主要研究方向为数据库技术、流数据处理、多维数据索引技术等.
E-mail:qianjiangbo@ nbu. edu. cn